In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set()
```

In [4]:
```
dataset = pd.read_csv('7458_Churn_Modelling.csv', index_col = 'RowNumber')
dataset.head()
```

Out[4]:

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
1	15634602	Hargrave	619	France	Female	42	2	(
2	15647311	Hill	608	Spain	Female	41	1	8380
3	15619304	Onio	502	France	Female	42	8	15966(
4	15701354	Boni	699	France	Female	39	1	(
5	15737888	Mitchell	850	Spain	Female	43	2	12551(

In [ ]:
```
#Customer ID and Surname would not be relevant as features
X_columns = dataset.columns.tolist()[2:12]
Y_columns = dataset.columns.tolist()[-1:]
print(X_columns)
print(Y_columns)
```

```
['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
'HasCrCard', 'IsActiveMember', 'EstimatedSalary']
['Exited']
```

In [ ]:
```
X = dataset[X_columns].values
Y = dataset[Y_columns].values
```

In [ ]:
```
#We need to encode categorical variables such as geography and gender
from sklearn.preprocessing import LabelEncoder
X_column_transformer = LabelEncoder()
X[:, 1] = X_column_transformer.fit_transform(X[:, 1])
```

In [ ]:
```
#Lets Encode gender now
X[:, 2] = X_column_transformer.fit_transform(X[:, 2])
```

We are treating countries with ordinal values(0 < 1 < 2) but they are incomparable. To solve this we can use one hot encoding. We will perform some standardization

In [ ]:
```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

pipeline = Pipeline(
```

```
    [
        ('Categorizer', ColumnTransformer(
            [
                ("Gender Label Encoder", OneHotEncoder(categories = 'auto', drop =
                ("Geography Label Encoder", OneHotEncoder(categories = 'auto', drop
            ],
            remainder = 'passthrough', n_jobs = 1)),
        ('Normalizer', StandardScaler())
    ]
)
```

In [ ]:
```
#Standardize the features
X = pipeline.fit_transform(X)
```

In [ ]:
```
#Spilt the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_s
```

In [ ]:
```
#Let us create the Neural Network
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

In [ ]:
```
#Initialize ANN
classifier = Sequential()
```

In [ ]:
```
#Add input layer and hidden layer
classifier.add(Dense(6, activation = 'relu', input_shape = (X_train.shape[1], )))
classifier.add(Dropout(rate = 0.1))
```

In [ ]:
```
#Add second layer
classifier.add(Dense(6, activation = 'relu'))
classifier.add(Dropout(rate = 0.1))
```

In [ ]:
```
#Add output layer
classifier.add(Dense(1, activation = 'sigmoid'))
```

In [ ]:
```
#Let us take a look at our network
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 6)                 72

 dropout (Dropout)           (None, 6)                 0

 dense_1 (Dense)             (None, 6)                 42

 dropout_1 (Dropout)         (None, 6)                 0

 dense_2 (Dense)             (None, 1)                 7

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
#Optimize the weights
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['ac
```

In [ ]:
```
#Fitting the Neural Network
history = classifier.fit(X_train, y_train, batch_size = 32, epochs = 200, validatio
```

In [ ]:
```
y_pred = classifier.predict(X_test)
print(y_pred[:5])
```

```
63/63 [==============================] - 0s 1ms/step
[[0.21353428]
 [0.3550975 ]
 [0.1884149 ]
 [0.04963601]
 [0.2057534 ]]
```

In [ ]:
```
#Let us use confusion matrix with cutoff value as 0.5
y_pred = (y_pred > 0.5).astype(int)
print(y_pred[:5])
```

```
[[0]
 [0]
 [0]
 [0]
 [0]]
```

In [ ]:
```
#Making the Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1569   26]
 [ 293  112]]
```

In [ ]:
```
#Accuracy of our NN
print(((cm[0][0] + cm[1][1])* 100) / len(y_test), '% of data was classified correct
```

```
84.05 % of data was classified correctly
```